EPL646 – Advanced Topics in Databases
Minibase

Christoforos Panayiotou
http://www.cs.ucy.ac.cy/~dzeina/courses/epl646/labs/lab.html

Overview

- History
- The Minibase Distribution
- minibase_globals
- Storage Manager
- Buffer Manager
- Heap File
- Error Protocol
- Declaring Errors
- Posting Errors
- Handling Errors

History

- Minibase is a database management system intended for educational use.
- It is based on Minirel, a small DBMS developed in UW, Madison.
- Evolved through many undergraduate and graduate course projects.

The Minibase Distribution

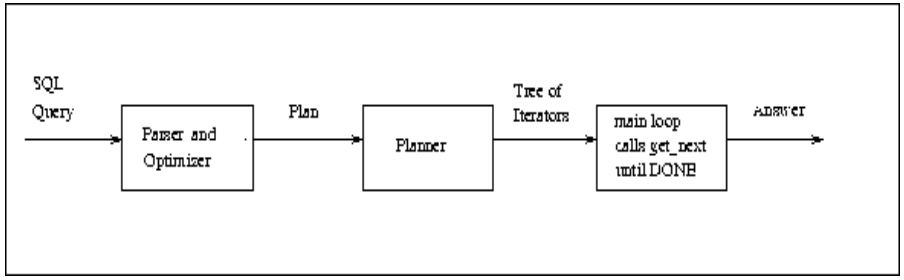
- It has a parser, optimizer, buffer pool manager, storage mechanisms (heap files, B+-trees as secondary indexes), and a disk space management system.
- Besides the DBMS, Minibase contains also a graphical user interface and other graphical tools for studying some DBMS issues.

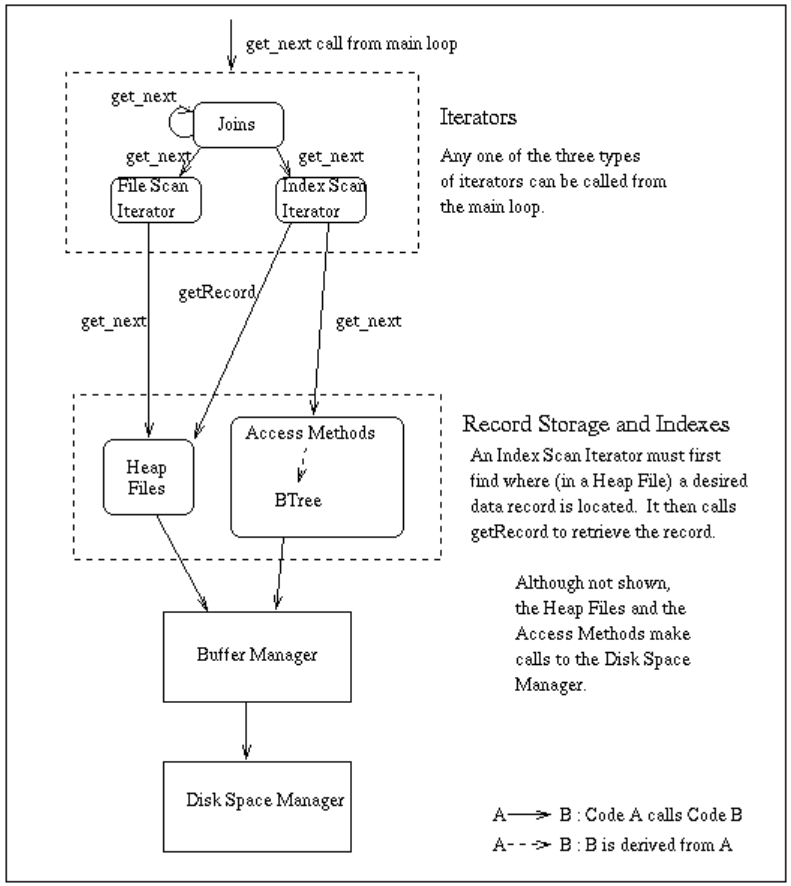
The Minibase Distribution (2)

Parser and Optimizer: These modules take an SQL query and find the best plan for evaluating it. The optimizer is similar to the one used in System R. In optimizing a query, the optimizer considers information in the catalog about relations and indexes. It can read catalog information from a Unix file.

Execution Planner: This module takes the plan tree produced by the optimizer, and creates a run-time data structure. This structure is essentially a tree of iterators. Execution is triggered by "pulling" on the root of the tree with a get-next-tuple() call.

The Minibase Distribution (3)





The Minibase Distribution (4)

Iterators: A “get-next-tuple” interface for file scans, index scans and joins.
Join Methods: Nested loops, sort-merge and hash joins are supported.
Heap Files: All data records are stored in heap files, which are files of unordered pages implemented on top of the DB class.
Access Methods: Currently only a single access method is supported, B+-trees. The access method in Minibase is dense, unclustered, and store key/rid-of-data-record pairs. Data records are always stored in heap files, as noted above, and access methods are implemented (like heap files) as files on top of the DB class.
Buffer Manager: The buffer manager swaps pages in and out of the (main memory) buffer pool in response to requests from access method and the heap file component.
Storage Manager: A database is a fixed size Unix file, and pages (in the file) on disk are managed by the storage manager.

minibase_globals

- The object **minibase_globals** is responsible for creating all its constituent objects to create or open a Minibase database, and for destroying them or to close it again. A database is opened by creating a SystemDefs object and assigning it to minibase_globals. A database is closed by deleting minibase_globals.

- The minibase_globals variable is a pointer to a SystemDefs object.

- The SystemDefs class has public data members for the various components of the system. These are referred to throughout the Minibase code by C preprocessor macros declared in system_defs.h (MINIBASE_BM for the buffer manager, MINIBASE_DB for the storage manager).

Storage Manager

- The Storage Manager is the component of Minibase that takes care of the allocation and deallocation of pages within a database. It also performs reads and writes of pages to and from disk, and provides a logical file layer within the context of a database management system.
- The abstraction of a page is provided by the Page class. Higher layers impose their own structure on pages simply by casting page pointers to their own record types. The data part of a page is guaranteed to start at the beginning of the block.
- The DB class provides the abstraction of a single database stored on disk. It shields the rest of the software from the fact that the database is implemented as a single Unix file. It provides methods for allocating additional pages (from the underlying Unix file) for use in the database and deallocating pages (which may then be re-used in response to subsequent allocation requests).

Buffer Manager

- **The Buffer Manager reads disk pages into a main memory page as needed.**
- The collection of main memory pages (called frames) used by the buffer manager for this purpose is called the buffer pool.
- The Buffer Manager is used by access methods, heap files, and relational operators to read / write /allocate / de-allocate pages.
- It makes calls to the underlying DB class object, which actually performs these functions on disk pages.
- Replacement policies can be changed easily at compile time.

Heap File

- A Heap File is an unordered set of records. The following operations are supported:
  - Heap files can be created and destroyed.
  - Existing heap files can be opened and closed.
  - Records can be inserted and deleted.
  - Records are uniquely identified by a record id (rid).

- The main kind of page structure used in the Heap File is HFPage, and this is viewed as a Page object by lower-level code. The HFPage class uses a slotted page structure with a slot directory that contains (slot offset, slot length) pairs. The page number and slot number are used together to uniquely identify a record.

Error Protocol

- Every subsystem creates error messages that describe the possible errors that will result.
- Each subsystem has its own set of error numbers.
- Each new error is added to the global queue.
- If the caller cannot recover from the error, it must append a new error to the global queue.

Declaring Errors

- **Error numbers**

  Provide an enumeration

  ```
  enum bufErrCodes { HASHTBLERROR,
                     HASHNOTFOUND,
                     BUFFEREXCEEDED, ... };
  ```

- **Error messages**

  ```
  static const char* bufErrMsgs[] = { "hash table error",
                                      "hash entry not found",
                                      "buffer pool full", ... };
  ```

- **Register errors**

  ```
  static error_string_table bufTable( BUFMGR, bufErrMsgs );
  ```

Posting Errors

- **Add first error**

```
MINIBASE_FIRST_ERROR( BUFMGR, BUFFEREXCEEDED );
```

- **Add chained error**

```
Status status = MINIBASE_DB->write_page( ... );
   if (status != OK )
         return MINIBASE_CHAIN_ERROR( BUFMGR, status );
```

- **Add first error produced by a previous one**

```
Status status = MINIBASE_DB->write_page( ... );
   if (status != OK )
         return MINIBASE_RESULTING_ERROR(BUFMGR, status,
BUFFEREXCEEDED );
```

Handling Errors

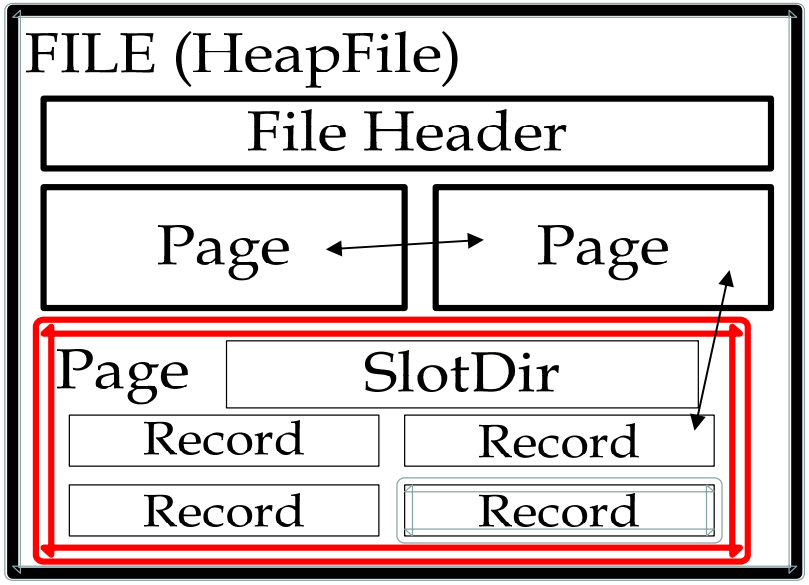
- Check the return value, which is of type Status, of a method.
- If OK, then no error was produced, otherwise handle error

More Information

Minibase Homepage

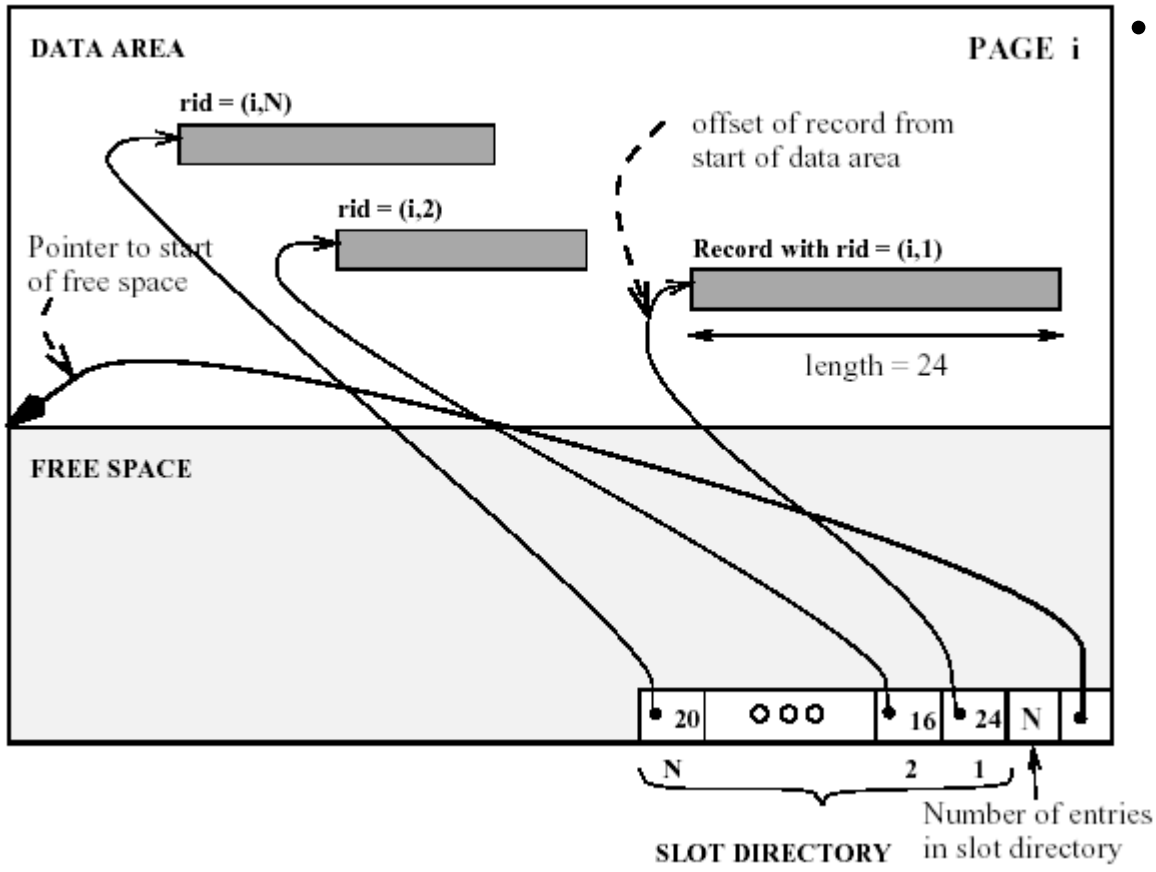
http://www.cs.wisc.edu/coral/minibase/minibase.html

Assignment 2



- Η δομή ενός Αρχείου Βάσης Δεδομένων το οποίο χρησιμοποιείται σε Αρχεία Σωρού (Heapfiles). Η διπλή γραμμή δείχνει το πλαίσιο αυτής της εργασίας το οποίο είναι η υλοποίηση της Σελίδας (HeapPage) ενός HeapFile και όχι του ιδίου του Heapfile.

Assignment 2:HFPage



- Δείτε το αρχείο hfpage.h του καταλόγου include/. Περιέχει τη διεπαφή (interface) για την κατηγορία HFPage. Αυτή η κατηγορία υλοποιεί ένα αντικείμενο «σελίδα αρχείου σωρού» (“heap-file page”). Σημειώστε ότι τα προστατευμένα πεδία μέλη (member fields) της σελίδας σας δίνονται. Το μόνο που πρέπει να πρέπει για να κάνετε είναι να υλοποιήσετε τις δημόσιες μεθόδους. Πρέπει να βάλετε τον κώδικά σας στο αρχείο src/hfpage.C.

Questions?

http://www.cs.ucy.ac.cy/~dzeina/courses/epl646/labs/lab.html